Title: Paper Airplanes and Python

Brief Overview:

The NCTM Standards 2000 state that "The computational capacity of technological tools extends the range of problems accessible to students and also enables them to execute routine procedures quickly and accurately, thus allowing more time for conceptualizing and modeling," (p. 25), and that "To understand the fundamentals of statistical ideas, students must work directly with data."

For this project, students first create paper airplanes, then measure the distance of 5 trials for each paper airplane, then write a computer program that accepts the input of 5 decimal (floating point) numbers and outputs the average of the 5 trials. Extensions are obvious (but also listed later in this document).

NCTM 2000 Principles for School Mathematics:

- **Equity:** *Students of all levels are motivated by hands-on projects. In this case, their making of paper airplanes in class is put to practical use.*

- **Technology:** *In order for students to be truly computer literate, they must be able to program computers to perform simple tasks to make the students' lives easier.*

Links to NCTM 2000 Standards:

- **Content Standards**

Number and Operations
Performing statistics in a programming environment requires that students understand the difference between integer division (with no remainder) and float division, and order of operations. Students should recognize when a mean varies wildly from the data set that a calculation was misperformed.

Algebra
A programming language will improve students' facility with formulae and variables.

Measurement
Students should be able to use nonstandard but convenient and consistent units of measurement, such as "number of brick lengths."

Data Analysis and Probability
Students write a computer program to find the mean, a measure of central tendency.

- **Process Standards**

  <u>Problem Solving</u>
  Students are motivated by writing computer programs; students receive instant feedback and are able to quickly and easily refine methods for solving a particular problem.

  <u>Reasoning and Proof</u>
  Programming requires students to formalize and linearize thought.

  <u>Communication</u>
  Programming is the act of communicating with a machine; good programming is the art of communicating with a machine in a manner understandable to other programmers.

  <u>Representation</u>
  Computer code is a representation of the students' mathematical thought.

Links to Maryland High School Mathematics Core Learning Units:

Functions and Algebra

- **1.1.1**
  Students will recognize, describe, and/or extend patterns and functional relationships that are expressed numerically, algebraically, and/or geometrically.

- **1.1.2**
  Students will represent patterns and/or functional relationships in a table, as a graph, and/or by mathematical expression.

- **1.1.3**
  Students will apply addition, subtraction, multiplication, and/or division of algebraic expressions to mathematical and real-world problems.

- **1.2.5**
  Students will apply formulas and/or use matrices (arrays of numbers) to solve real-world problems.

Geometry, Measurement, and Reasoning

- **2.1.3**
  Students will use transformations to move figures, create designs, and/or demonstrate geometric properties.

- **2.2.3**
  Students will use inductive or deductive reasoning.

Data Analysis and Probability

- **3.1.1**
  Students will design and/or conduct an investigation that uses statistical methods to analyze data and communicate results.

- **3.1.2**
  Students will use the measures of central tendency and/or variability to make informed conclusions.

- **3.2.3**
  Students will communicate the use and misuse of statistics.

Grade/Level:

Although our students were grades 9-12, this activity could easily be adapted to any grade level.

Duration/Length:

30 minutes collecting data; 15 minutes chalk-talk; approx. 45 minutes coding / testing / revising programs.

Prerequisite Knowledge:

Students should have working knowledge of the following skills:

- Basic competence with the given programming environment (e.g., Python).
- This is a first lesson for the concept of functions. Students should be ready for functions.
- Students should be able to compute the mean of a fixed-length data set.

Student Outcomes:

Students will:

- write a program that uses functions to perform repetitive or computational tasks.

Materials/Resources/Printed Materials:

- Handouts (attached)
- Programming environment (e.g., Python with the IDLE environment, freely available from www.python.org. This unit could be adapted to anything from calculator BASIC to C.)
- Paper for paper airplanes

Development/Procedures:

- Have students read worksheet; and discuss and explain procedure, timing expectations, etc.

- Students create paper airplanes. They should be attempting to maximize distance.

- Class makes field trip to appropriate location for throwing paper airplanes. We used low steps above a brick courtyard, and measured distance in terms of bricks. Each student was responsible for measuring her / his airplane, after the instructor marked the spot where the plane landed. Either of these tasks could be delegated to a team of specialized students.

- Return inside and discuss appropriate program structure. We stressed that each program should start with a heading indicating author and program name and description; followed by an input block, a computation block, and an output block.

- Now that we are using functions, instead of using blocks for each of those tasks, we use functions that perform those tasks. We compared the following prompts:

    - "What was the distance in bricks for trial number 1? "
    - "What was the distance in bricks for trial number 2? "

- Students were able to recognize that everything was the same for the two prompts except for the trial number. This made trial number an appropriate function PARAMETER. The input function should be declared "`def askuser(trial):`"
  and since what the user entered should be passed back to the main function of the program, it should include the line "`return answer`".

- The computation function was given the header "`def average(num1, num2, num3, num4, num5):`" and the output function, which in other languages would be called a procedure or void, was given the header "`def output(result):`"

- The perhaps awkward "`askuser`" was used because there is a command "`input`"; the parameter "`result`" was used with "`output`" in order to avoid confusion with the function name "`average`".

- Because this was one of the first times we used functions, we also discussed what the main part of the program should look like. Students with prior programming experience tended to use a proper main function, that would be called from the `*Python shell*` window by the prompt

  ```
  >>> programname.main( )
  ```

Novices have the following outline:

```
def askuser(trial):

def average(num1,num2,num3,num4,num5):

def output(result):

#main part of program
num1 = askuser(1)
num2 = askuser(2)
num3 = askuser(3)
. . .

result = average(num1,num2,...)

output(result)
```

- If we did this project later in the year, we would expect the conventional but initially intimidating

```
if __name__ == '__main__':
    main()
```

- Most students used the following code for the input function:

```
def askuser(trial):
    answer = float(raw_input("what is the distance for trial
"+'trial'+"?  "))
    return answer
```

- Although Python has an "input" function that will accurately guess whether the user input is an int or float, it is not considered safe programming practice to use this function, as the user could also enter commands that would be executed by the interpreter. "raw_input" always returns a string, which the program needs to convert to a numeric type.

- Students who converted the input to an int and also failed to force the result of the averaging function to be a float experienced runtime errors when the average of five numbers was a non-whole number.

- Finally, the most common initial error was forgetting the order of operations for the line

```
    result = (num1 + num2 + num3 + num4 + num5) / 5.
```

  the parentheses enforce correct order of operations, where the decimal point on the 5. ensures that the result will be a floating point number.

Assessment:

Computer programs can be assessed on a 4 point rubric (as with an Extended Constructed Response item from the Maryland High School Assessments), with the understanding that points do not map directly to percentages.

4 points / A:

The program:

runs correctly (has appropriate output for given inputs),

AND

is divided into appropriate functions,

AND

is documented and / or uses self-documenting variable names,

AND

performs error checking on user inputs.

3 Points / B:

The program:

runs correctly for expected inputs, but

crashes when user gives unexpected or inappropriate input

XOR

is monolithic (not divided into functions)

XOR

uses neither self-documenting variable and function names nor appropriate other documentation

2 Points / C:

The program:

does not crash, but gives output that is not appropriate to the user inputs

OR

has multiple deficiencies as listed for a 3 point program

1 Point / D:

The program:

Fails to run, but reflects some sort of algorithm appropriate to the given task

0 Points /E/F:

The program

Fails to run

AND

Does not reflect any sort of algorithm appropriate to the task at hand.

Extension/Follow Up:

This activity can be extended as follows:

- When students know lists, the program can evaluate the mean of a variable-length set of data; either with a for loop (if the user states at the outset how many trials there were) or with a while loop (if the user enters an escape value such as -1 to indicate that s/he is done entering data)
- The program could also calculate other measures of central tendency and the standard deviation of the set of data; this is an appropriate in-class extension for students who finish more quickly than others
- If both of the above extensions are combined, students could create a general purpose, numerical menu-driven statistical package.

- Another program that would use many more computational functions, and that would probably call computational functions from within other computational functions, is the attached split-time activity. The user should be able to enter in lap times in minutes and seconds, and receive an output of how long each lap took. In order to collect data for this, we had students walk around the oval at UMES three times; any circuit will do. In order to complete this project students need to convert times in minutes and seconds to times in seconds and back again (1:30 - 0:50! = 0:80). Students should be familiar with the modulo or "remainder" % operation.

Authors:

Lloyd Hugh Allen
Parkville High School and Center for Math, Science, and Technology
Baltimore County, MD

Tim Gavin
Parkville High School and Center for Math, Science, and Technology
Baltimore County, MD

Flight Length

Name_____

Directions: Read all directions first. The Boeing Jet Company has hired you to analyze the glide path of certain planes. Your first task is to compile length of flight information on various types of gliders. You will find the mean flight time for your glider. This has to be an automated process due to the overwhelming amount of flight data that will be produced. We will simulate the flights by making our own flights.

Step 1: Construct a paper airplane of suitable structure. Remember that it should fly for length.

Step 2: Test airplane outside.

Step 3: Redesign or modify plane.

Step 4: Hold flight trials and gather data.

Data table:

Flight Number	Distance
1	_____
2	_____
3	_____
4	_____
5	_____

Step 6: Write a Python Program that asks the user to input the five pieces of data and then computes the mean of the data. Your program should include a function that computes the average. This function should be called from within your program.

Running the Race

Name_____

Directions: Read all directions first. You are going to write a computer stopwatch program to help coaches analyze their runner's times. A coach finds it useful to know a runner's split times for races, that is, the time it takes a runner to complete each part of the race. Unfortunately for us, the coach just takes cumulative times for his runners. We have to find the split times. This activity has two parts, data collection, and programming. To collect the data, you will be walking around the quad three times. We will record your cumulative time for each lap. You will then write a program that will ask for the time after each lap and then compute and output the split time.

Example:

Input

Fast Fred's cumulative times:

	Lap 1	5:16
	Lap 2	10:40
	Lap 3	16: 30

Output

Fast Fred's split times	Lap 1 split:	5:16
	Lap 2 split	5:24
	Lap 3 split	5:50
	Total	16:30

You will have a partner for this activity. Please record their cumulative times on your sheet and use that for you input.

Name of Partner: _____

Lap 1: _____

Lap 2: _____

Lap 3: _____

Special Notes to keep in Mind:

1. Keep all times in minutes and seconds.
2. You can't just subtract times.
3. Document your code.